

DATA PROCESSING SYSTEMS AND METHOD FOR PROCESSING
WORK ITEMS IN SUCH SYSTEMS

Field of the Invention

5 The present invention relates generally to processing of work items in data processing systems and more particularly to the scheduling of tasks to process such work items.

10 Background of the Invention

15 In modern data processing systems there are two well established methods for the communication required between hardware and software for software control of hardware devices within the system: interrupts and polling. With
20 interrupts, each hardware device signals that there is work for the software to do by asserting an interrupt line which causes the software flow-of-control to be diverted to an interrupt handler which handles the interrupt. An interrupt
25 is typically handled by masking (or disabling) the interrupt and scheduling a task for later execution which will service the requesting device. Once the device has been serviced, the interrupt is unmasked (or enabled) again ready to allow the generation of further interrupts. With polling, the device driver continually checks the status of the device to determine whether it needs to be serviced. If so, the device is serviced.

With polling based systems, there is a trade-off between polling frequently so as to service the device with low latency and polling infrequently so as to keep the overhead of polling to a minimum. With interrupt-based systems, the device is serviced with low latency and low overhead when it is relatively idle but when the device becomes heavily utilised and is generating many interrupts the increased interrupt overhead can make the interrupt method inefficient. Generally speaking therefore, an interrupt based system is more efficient at relatively low device utilisation and a polling based system is more efficient at relatively high device utilisation. Neither technique is especially well adapted to the situation where the device presents varying amounts of work to the system.

Attempts have been made in the art to reduce the interrupt overhead in interrupt-based systems. In IBM Technical Disclosure Bulletin Vol 38 No 7 pp 497-500, a technique is described that is designed to reduce the number of interrupts generated by a LAN adapter. One type of interrupt (transmit complete) which does not impact response time is turned off thus reducing system disruption and CPU utilisation. An alternative method for identifying and reporting transmit completions is provided. Although the described technique goes some way to reducing interrupt overhead in the described environment, it necessitates the provision of an alternative method to handle the disabled interrupt.

It would be desirable to have a system having low latency at low workload levels and low overhead at high workload levels.

5

Disclosure of the Invention

According to a first aspect of the invention there is provided a method for processing work items in a data processing system comprising: generating an interrupt in response to receipt of a work item in the system; servicing the generated interrupt to schedule a task for later processing of the work item, without re-enabling the interrupt; subsequently executing the task to process the work item; and speculatively scheduling a further task for processing of any work items that are subsequently received in the system.

The present invention is thus contrasted with prior systems in which, when an interrupt is serviced and a task scheduled for later execution, the interrupt is enabled/unmasked. Thus in prior systems, the receipt of further work items will cause the generation of further interrupts. In the present invention, the interrupt is not enabled when the interrupt is serviced and therefore further work items will not generate interrupts. When the task executes, it processes the work items after which a

speculative task is scheduled and added to the task queue.
The task is speculative in the sense that when it is
generated there are no work items on the queue to be
processed. However it is anticipated that further work items
5 will have been added by the time the speculative task
reaches the head of the task queue and is executed.

In accordance with one embodiment of the invention, the
method could include the step of continually scheduling
10 speculative tasks (i.e. polling) for processing of work
items that may subsequently be received in the system.
However this embodiment, which effectively comprises the use
of a single interrupt followed thereafter by polling, is
only suitable for systems which have a device which is
either never used or is always heavily used during the
period of time between system resets.

In a preferred method, when the speculatively scheduled
task is executed to process any work items received by the
system and it is determined that there are no work items,
the interrupt is enabled. Thus when the system is fully
utilised, the interrupt mechanism is replaced with a polling
mechanism involving a continuous series of speculatively
scheduled tasks. However when the system or device
20 utilisation decreases, i.e. when there are no work items
when the speculatively scheduled task is processed, then the
system reverts to interrupts.
25

In one alternative embodiment, when device presents a work item to the system, the device operation may stall pending processing of the work item by the scheduled task. This embodiment may provide a benefit in terms of overall system performance in the case where the system throughput is limited by processor utilisation because the interrupt overhead would be eliminated. In a preferred embodiment however, a work item queue is maintained for the interrupting device so that the operation of the device is not halted when the interrupt is disabled.

According to another aspect of the invention there is provided a data processing system comprising: processing means for executing tasks to process work items in the data processing system; and interrupt generating means for generating an interrupt in response to receipt of a work item in the system; wherein the processing means is operable to: service the generated interrupt to schedule a task for later processing of the work item, without re-enabling the interrupt; subsequently execute the task to process the work item; and speculatively schedule a further task for processing of any work items that are subsequently received in the system.

According to a further aspect of the invention there is provided a computer program product comprising a computer usable medium having computer readable program code means embodied in the medium for processing work items in a data

processing system, the program code means comprising: code means for causing the data processing system to service a generated work item interrupt to schedule a task for later processing of the work item, without re-enabling the interrupt; code means for causing the data processing system to subsequently execute the task to process the work item; and code means for causing the data processing system to speculatively schedule a further task for processing of any work items that are subsequently received in the system.

The computer program product according to the further aspect of the invention may be implemented in a read-only memory, magnetic tape or diskette.

The present invention may advantageously be used to reduce overall latency in any interrupt based data processing system where it is expected that there will be a delay between scheduling and execution of tasks.

Brief Description of the Drawings

A preferred embodiment of the present invention will now be described with reference to the accompanying drawings in which:

Figure 1 is a schematic representation of a processing apparatus embodying the invention;

Figure 2 is a flow diagram showing the steps involved in a method according to a preferred embodiment of the invention; and

5 Figs 3A, 3B and 3C are schematic representations of the state of the task and work item queues of the preferred embodiment of the invention at different stages of the method of Figure 2.

10 Description of the Preferred Embodiments

In the following the present invention will be described in relation to a storage controller (e.g. storage adapter card) receiving work items from an attached host system. It will be appreciated that this is by way of example only -- the invention may be used in many suitable systems.

15
20
25 With reference first to Figure 1, there is shown a processing apparatus comprising a host system 10, which may be a personal computer system, workstation, mainframe or the like, for example an IBM RS/6000 system running the AIX operating system. The host system includes a device driver 20 and a gateway 30 for providing communication between the device driver and an attached storage controller 100. In the present embodiment, the storage controller takes the form of an adapter card which plugs into a bus (for example a PCI bus) within the host system. The storage controller is in

turn attached for communication with a storage subsystem 200 that may comprise an array of data storage disk devices.

5 The storage controller includes a gateway 102 for communicating with the host system, a microprocessor 104 and DRAM 106 for storing data and metadata, including data being transferred between the host and storage subsystem. Also provided as part of the storage controller is ROM 108 in which is stored firmware that executes on the microprocessor
10 to provide a variety of different services within the controller. The services take the form of a variety of different tasks which are scheduled on a task queue for execution by the microprocessor. One service provided by the firmware is the processing of requests from the host system to read and write data from and to the storage subsystem. Each such data transfer request will generally involve a number of different firmware tasks which during the operation of the controller are placed on the task queue within the DRAM. Disk interface 112 provides the
20 communication interface to the storage subsystem. The controller further includes data flow manager 110 that is adapted to control the data flow within the controller. As will be described below, the data flow manager receives work items from the host system and generates interrupts for
25 servicing by the controller firmware.

Next will be described, in general terms, the operation of the processing apparatus of Figure 1. In response to host

application requests for data, the host system device driver generates host transactions (otherwise referred to herein as work items) for processing by the storage controller. The work items are transferred via the gateway to the storage controller and placed on a work item queue in DRAM by the data flow manager. In response to receipt of each work item, a task is caused to be placed on a task queue in DRAM for subsequent execution by the microprocessor. To take the example of a disk write operation, the appropriate task is placed on the task queue for subsequent execution on the processor to DMA the write data from the host system memory into the controller DRAM. It should be noted that tasks other than those involved in processing host work items will also be placed on the task queue for execution by the microprocessor. In the present embodiment, the task queue is managed by the microprocessor as a FIFO queue in a manner well known in the art, with tasks added to the tail of the queue.

In prior systems of this type which are interrupt driven, the receipt of a work item at the controller will result in the generation of an interrupt to the microprocessor. As discussed previously, such a system provides low latency and low overhead provided that the rate of interrupts is low. If however, the number of work items from the host is significantly increased, the interrupt overhead will be increased to a level which may impact the overall operation of the controller microprocessor.

With reference next to Figure 2 and Figures 3A to 3C, there will be described a process, operable on the storage controller of Figure 1, for processing work items from the host system in a manner that combines the best attributes of the polling and interrupt methods to service the host work items with low latency at low utilisation and by polling for low overhead at high utilisation.

In accordance with the present embodiment, the process starts at step 300 where the controller is initialised. At step 310, the host work item interrupt is enabled ready to allow the generation of a host work item interrupt in response to receipt of a work item at the controller. At step 320, a host work item is received by the controller data flow manager and placed on the host work item queue in DRAM. As the interrupt is enabled, the data flow manager generates an interrupt to the microprocessor which causes the microprocessor to halt its operation and to load the appropriate interrupt handler code from ROM. At step 330, the microprocessor disables the interrupt thereby preventing the generation of any host interrupts pending processing of the work item. At step 340, the interrupt handler schedules a task for processing the work item by placing the task on a FIFO task queue within the DRAM. The interrupt handler then completes and the processor returns to the operation it was carrying out prior to the interruption. In contrast with prior systems, the interrupt is not enabled at this point;

this means that if and when a further work item is received on the work item queue, an interrupt is not generated. The status of the work item and task queues at this stage of the process can be seen in Figure 3A. The work item queue includes one item I1 and the task queue includes a number of tasks including T1 at the tail of the task queue.

Unless it is subject to other interruptions from other devices, the microprocessor continues by processing the tasks on the queue and eventually task T1 reaches the head of the queue. During the period between when task T1 was placed on the queue and when T1 reached the head of the queue, three further work items have been queued by the data flow manager. As the interrupt was disabled, no interrupts were generated in response to receipt of these work items. The status of the work item and task queues at this stage of the process can be seen in Figure 3B in which the work item queue includes three items I1 to I3 and the task queue includes a number of tasks including task T1 at the head of the queue.

At step 350 the scheduled task T1 is loaded and executed by the microprocessor to process the work item I1 (step 360). At step 370, the processor checks for further work items on the work item queue and on finding item I2, processes that as well. The processor continues to check for and process further work items until the queue is empty.

In accordance with the embodiment of the invention, on finding the queue to be empty, the processor, at step 380, causes a further task for processing host work items to be scheduled by placing it on the task queue. The interrupt is not enabled/unmasked. The status of the work item and task queues at this stage of the process is as indicated in Figure 3C in which the work item queue is empty and the task queue includes a number of tasks including the speculatively scheduled task at the tail of the queue.

This further task is speculative in nature in the sense that the processor is anticipating that further work items will appear on the queue in the meantime and therefore when the speculative task reaches the head of the queue, it will have work items to process. Because the interrupt is not enabled when the speculative task was scheduled, none of the work items which are subsequently added to the queue will generate interrupts, thus avoiding interrupt overhead.

Again assuming that no further interrupts are received, the microprocessor continues by processing the tasks on the queue until task T2 reaches the head of the queue. At step 390, the microprocessor executes task T2 which determines (step 400) whether there are any work items on the work item queue. If yes, the task processes the queued work items at step 410. On a determination that there are no further work items, the processor causes a further speculative task to be scheduled (step 380) and placed on the task queue. Although

this scheduling of a speculative task involves a certain amount of overhead, this is worthwhile in view of the likelihood that the work items will continue to be placed on the queue.

5

If at step 400 it is determined that there are no host work items on the queue, then the process moves to step 310 and the interrupt is enabled/unmasked. Thus when no further work items have been added to the queue in the time between the task T2 is added to the task queue and when it reaches the head of the queue, the host and/or controller workload is low and the process reverts to an interrupt driven process for the host work item interrupts.

10

Thus it will be noted that when the controller is relatively busy (i.e. when there are many different tasks on the task queue), the speculatively scheduled task will be delayed by other tasks which are currently in progress and thus the length of the delay will increase according to how busy the controller is. During the period the speculative task is delayed, work items from the host will be added to the queue. Because the interrupt is disabled/masked, none of these work items results in the generation of an interrupt and therefore the interrupt overhead which would occur in prior systems is avoided. When the speculative task reaches the head of the queue all the pending work items are processed.

20

25

When both the arrival rate of host work items and the controller utilisation are low, the pattern of operation is that an arriving host work item generates an interrupt which schedules a task to service the work item. The task
5 services the work item and a speculative task is scheduled. The speculatively scheduled task is executed a relatively short time after the interrupt scheduled task because there are few other outstanding tasks to execute on the controller between the interrupt scheduled task and the speculatively
10 scheduled task. The short delay between the interrupt scheduled task and the speculatively scheduled task and the low arrival rate of host work items means that the speculatively scheduled task does not find host work items waiting when it polls for them so the speculative polling process is terminated and the interrupt mechanism is
15 restored after only one speculatively scheduled task. Thus the host system is serviced with the low latency of the interrupt method with some additional overhead caused by the speculatively scheduled tasks. This additional overhead
20 does not impact the overall operation of the controller because the system is relatively idle and the controller is under-utilised.

As the arrival rate of host work items increases and/or
25 the controller utilisation increases (due for instance to an increased arrival rate of work items from a different source), there is more opportunity for host work items to arrive between the interrupt scheduled task and the

speculatively scheduled task (or between a speculatively
scheduled task which found host work items and the
speculatively scheduled task which was scheduled as a
result) so the speculatively scheduled task will more
frequently find host work items waiting and the speculative
polling process will go on for longer before the interrupt
mechanism is restored. Thus the number of interrupts
avoided and the size of the 'batches' of work items
processed by each task increases both with the arrival rate
of host work items and with the utilisation of the
controller.

Although on the face of it, it might appear that by use
of the technique described above, the delays in processing
the work items might increase the latency of the system at
high utilisations over that of the conventional interrupt
method, this is not in fact the case. The reduced overhead
of this technique more than compensates by keeping the queue
depth down. In the limit, the present invention allows the
system to operate with relatively low latency at throughputs
beyond the limit where the conventional interrupt method
would have caused the queue depth and latency to tend to
infinity.

While an embodiment of the invention have been
described in detail above, it will be apparent to those
skilled in the art that many variations and modifications
can be made without departing from the scope of the

invention. In particular, although in the described embodiment, the queued work items were host transactions, it will be appreciated that the use of the invention in a storage controller is not limited to such operations. For example, the technique of the present invention may be used with other interrupt-generating events such as DMA completions and messages received from an attached storage controller. Multiple queues would be maintained in the event that the technique of the present invention is used in relation to multiple events.

Furthermore, although the invention has been described in relation to a storage controller, it will be apparent that the invention may be used in any data processing system in which interrupts are generated in response to the receipt of work items from a device.

In addition, although the work items are described as being managed in a queue, in alternative embodiments, the work items may be arranged as an array where a work item exists in a fixed location and is marked as requiring servicing by the hardware setting a bit in the array.